CRT-2001-09

Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

COOPERATIVE PARALLEL TABU SEARCH FOR CAPACITATED NETWORK DESIGN

by

Teodor Gabriel Crainic
Michel Gendreau

Mars 2001

102-02899/

PUBLICATION
CRT-2001-09

*COOPERATIVE PARALLEL TABU SEARCH FOR CAPACITATED NETWORK DESIGN**

by

Teodor Gabriel Crainic[1]
Michel Gendreau[2]

* This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

1 Département de management et technologie, Université du Québec à Montréal, et Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7

2 Département d'informatique et de recherche opérationnelle et Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2001

Abstract

We present a cooperative parallel tabu search method for the fixed charge, capacitated, multicommodity network design problem. Several communication strategies are analyzed and compared. The resulting parallel procedure displays excellent performances in terms of solution quality and solution times. The experiments show that parallel implementations find better solutions than sequential ones. They also show that, when properly designed and implemented, cooperative search outperforms independent search strategies, at least on the class of problems of interest here.

Key words: Service Network Design, Parallel Tabu Search, Cooperative Search

Résumé

Nous présentons une méthode de recherche tabou parallèle coopérative pour le problème de synthèse de réseaux multiproduits avec coûts fixes et capacités. Nous analysons plusieurs stratégies de communications et d'échange d'information. Les résultats expérimentaux montrent que la procédure parallèle offre de très bonnes performances en termes de qualité de solution et de temps de résolution. Ces mêmes résultats démontrent également que les recherches coopératives offrent de meilleures performances que les procédures de recherche indépendantes, du moins pour les classes de problèmes étudiées dans cet article.

Mots-clés : Synthèse des réseaux de service, recherches avec tabous, parallélisme, recherches coopératives.

1 Introduction

The fixed charge network design problem is a well-known problem, of both practical and theoretical significance. Fixed charge capacitated multicommodity network design formulations with linear costs (*CMND*) appear prominently when addressing issues in infrastructure or service network construction or improvement. Transportation, logistics, telecommunication, as well as power system design and production planning are among the main application areas of this class of formulations. Here, several "commodities" – goods, data packets, people, ... – have to be moved, from their respective origins to their particular destinations, over the links of a network with limited capacities. Two types of costs are incurred. A *variable* "transportation" costs related to the volume of each commodity flowing though a given link, and a *fixed* construction or utilization cost that is payed as soon as a link is used or capacity is added.

Most network design formulations present considerable algorithmic challenges, especially when one attempts to solve realistically sized problem instances. These usually take the form of mixed-integer optimization models with multicommodity network flow constraints and a number of complicated additional restrictions. The mathematical formulations are combinatorial and, usually, NP-hard. The trade-offs between the variable and fixed costs inherent in the selection of any given solution, as well as the interplay between the limited network capacity, the resulting competition among commodities, and the fixed costs associated using the links of the network, add to the difficulty to efficiently solve CMND problems. For details on network design formulations, solution methods, applications, and recent results, the interested reader may consult Magnanti and Wong (1986), Minoux (1986), Balakrishnan, Magnanti, and Wong (1989), Balakrishnan, Magnanti, and Mirchandani (1997), Gendron and Crainic (1994, 1996), Crainic, Frangioni, and Gendron (1998) Gendron, Crainic, and Frangioni (1998), Crainic (2000, 1999), and the references cited in these papers.

Metaheuristics and parallel computation appear as important building blocks of efficient solution methods for complex problems. Metaheuristics, and particularly tabu search procedures (Glover, 1989, 1990; Glover and Laguna, 1997), are increasingly used to identify good feasible solutions to realistically sized network design problems. Thus, Crainic, Gendreau, and Farvolden (2000) have proposed a tabu search metaheuristic that currently appears to be the best procedure for finding good feasible solutions for the *CMND*. Parallel computing offers the possibility to design procedures that exploit more efficiently the solution space. This may be achieved through the acceleration of some tedious computational phases of the algorithm, the decomposition of the problem domain or search space, or the design of a multi-thread parallel search with various degrees of synchronization and cooperation. It also opens the door to efficient combinations of heuristic procedures and exact algorithms, such as branch-and-bound. See, for example, Gendron and Crainic (1994), Grama and Kumar (1995), Crainic, Toulouse, and Gendreau (1997), Crainic and Toulouse (1998), and the references cited in these papers.

We develop and analyze a parallel cooperative multi-search method for the *CMND*. This approach appeared in previous studies (e.g., Crainic, Toulouse, and Gendreau, 1995b) as the most appropriate for formulations which, similarly to the network design problems we address, combine a combinatorial nature to complex evaluation subproblems (in the present case, a capacitated multicommodity minimum flow problem). The parallel approach is based on several tabu search threads which cooperate by asynchronously exchanging information about the best solutions identified so far. The individual searches use the same sequential tabu search method (Crainic, Gendreau, and Farvolden, 2000), but differ in the values of a number of search parameters and, eventually, initial solutions. The results that we present show that the parallel procedure achieves both excellent quality solutions and remarkable accelerations as compared to its sequential counterpart.

The main contributions of this paper are as follows. First, we present several cooperation strategies – what information to exchange and when to exchange it – and analyze their impact on the search trajectory and the solutions reached. In particular, we present and compare several memory-based strategies for the selection of the information to exchange among individual search threads. This contributes towards a better understanding of parallel tabu search methods. Second, we describe a parallel procedure which displays excellent performance measures and which allows to solve efficiently large capacitated fixed charge multicommodity network design problems.

The paper is organized as follows. Section 2 recalls the formulation of the problem and the basic ideas of the tabu search method. The parallel strategies are detailed in Section 3. Experimental results are reported and analyzed in Section 4. We conclude with a few general remarks and research perspectives.

2 Formulation and Sequential Tabu Search

This section is dedicated to a recapitulation of the general formulation and the sequential tabu search procedure. Crainic, Gendreau, and Farvolden (2000) present full descriptions and analyses.

The tabu search procedure is developed for the path–based formulation of the fixed-cost, capacitated multicommodity network design problem (*PCMND*). The model is defined on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ with $\mathcal{N}$ the set of nodes, $\mathcal{A} = \{a = (i,j) \,|\, i \neq j, i \in \mathcal{N},\ j \in \mathcal{N}\}$ the set of arcs (for the sake of simplicity, we assume all links to be directed design arcs), and $\mathcal{P}$ the set of commodities to be distributed. Without loss of generality, we define a commodity $p \in \mathcal{P}$ to be the flow between an origin node $r^p \in \mathcal{N}$ and a destination node $s^p \in \mathcal{N}$, $r^p \neq s^p$, while the corresponding demand is denoted w^p.

Three measures characterize each arc $a \in \mathcal{A}$: c_a^p, the unit cost of moving commodity

$p \in \mathcal{P}$ through the link, f_a, the fixed cost of including the arc in the final design of the network (or to introduce additional capacity on the arc), and u_a, the total capacity of the arc. The flow of commodity p moves from its origin r^p to its destination s^p by using one or several paths in $\mathcal{L}^p$. According to standard network notation, $\delta^p_{al} = 1$ if arc a belongs to the l^{th} path for commodity p and 0 otherwise, while the unit transportation cost of path $l \in \mathcal{L}^p$ is $k^p_l = \sum_{a \in \mathcal{A}} c^p_a \delta^p_{al}$.

The *PCMND* formulation may then be written as follows:

$$\text{Minimize} \quad z(h,y) = \sum_{a \in \mathcal{A}} f_a y_a + \sum_{p \in \mathcal{P}} \sum_{l \in \mathcal{L}^p} k^p_l h^p_l \tag{1}$$

$$\text{subject to} \quad \sum_{l \in \mathcal{L}^p} h^p_l = w^p \qquad p \in \mathcal{P} \tag{2}$$

$$\sum_{p \in \mathcal{P}} \sum_{l \in \mathcal{L}^p} h^p_l \delta^p_{al} \leq u_a y_a \qquad a \in \mathcal{A} \tag{3}$$

$$h^p_l \geq 0 \qquad p \in \mathcal{P},\ l \in \mathcal{L}^p \tag{4}$$

$$y_a \in \{0,1\} \qquad a \in \mathcal{A} \tag{5}$$

where $y_a = 1$ if arc $a \in \mathcal{A}$ is included (opened) in the final design of the network and 0 otherwise, and h^p_l represents the flow of commodity $p \in \mathcal{P}$ on path $l \in \mathcal{L}^p$. The corresponding arc flows, x^p_a, may be obtained by the usual computation $x^p_a = \sum_{l \in \mathcal{L}^p} h^p_l \delta^p_{al}$.

The sequential tabu search method is based on the exploration of the space of extreme points $\tilde{\mathcal{H}}$ of $\mathcal{H}$ defined by equations (2), (3), and (4). The adjacency relationships in $\tilde{\mathcal{H}}$ define a natural neighborhood structure, while the pivoting rules of the simplex method provide an efficient way to determine the neighbors of any given solution. Furthermore, for any given path flow pattern $\tilde{h} \in \tilde{\mathcal{H}}$, a design with minimal cost relative to the path flows, denoted $y(\tilde{h})$, can be obtained by setting

$$y_a(\tilde{h}) = \begin{cases} 0 & \text{if } \sum_{p \in \mathcal{P}} \sum_{l \in \mathcal{L}^p} \tilde{h}^p_a \delta^p_{al} = 0\ , \\ 1 & \text{otherwise.} \end{cases}$$

The algorithm, displayed in Figure 1, exhibits the main structure of the primal simplex method with basis partitioning and column generation, similar to that of Farvolden, Powell, and Lustig (1992). Following an initialization phase, the search proceeds with a sequence of local searches and diversification phases. Each local search is made up of several alternating series of pivot moves and path generation (neighborhood expansion) phases. Computations stop with the best solution encountered during the search, Z_{best}, once a predefined number of diversification phases, *max_div*, have been performed.

Formally, $\mathcal{N}(\tilde{h})$, the continuous (local search) *neighborhood* of an element $\tilde{h} \in \tilde{\mathcal{H}}$, is defined as the subset of all extreme points of $\tilde{\mathcal{H}}$ adjacent to $\tilde{h}$, that is, all extreme points which can be reached from $\tilde{h}$ by one simplex pivot. In Linear Programming terms

1. *Initialization*;
   C: Set of candidates – current paths;
   $Z_{best} = \infty$; $Z_{local} = \infty$; $div = 0$.

   Local Search (Steps 2 to 10)

2. *Initialize* short term tabu lists; $Z_{prev} = Z_{local}$; $g = 0$.

3. *Neighbor evaluation and move selection:*
   Repeat steps 4 to 8 until *max_move* consecutive pivots (moves) have been executed with no improvement in the value of Z_{local}.

4. Compute the values of the dual variables;
   Pivot in any slack variable with negative marginal cost;
   Compute the reduced costs for all non-basic paths.

5. For each non basic path variable

   - Determine the corresponding exiting (basic) path variable;
   - Compute the value of the potential move.

6. Identify the best move.

7. *Pivot–move:* If the selected move is not tabu, implement it;
   If move is tabu, implement only if it improves Z_{best};
   Otherwise, select next best candidate move.

8. Establish and update tabu tenures;
   Update long term memories;
   If improved, update Z_{local} and, eventually, Z_{best}.

   Column generation

9. If $Z_{local} < Z_{prev}$ then $Z_{prev} = Z_{local}$ and $g = 0$;
   Else $g = g + 1$ and if $g >$ *max_col_gen* Go to Step 11.

10. Compute the values of the dual variables using the surrogate costs;
    Generate new paths and add them to C; Go to Step 3.

11. Diversification: If $div <$ *max_div* then
    Perform diversification moves;
    Set Z_{local} to $z(h, y)$ after diversification;
    $div = div + 1$; Go to Step 2

12. Else STOP

Figure 1: Sequential Tabu Search Procedure

(Dantzig and Thapa, 1997), a *move* thus corresponds to a transition from a basis of the system defined by equations (2) to (4) to an adjacent one. A basic path thus becomes non-basic, while a previously non-basic path takes its place in the basis.

A *candidate move* is then simply a pair of variables (*path_in_basis*, *path_not_in_basis*) on which a simplex pivot may be performed. The *candidate list* is the set of all such pairs of variables. To avoid cycling, recently pivoted out variables are forbidden to enter back into the basis for a number of moves. The *tabu tenure* of each exiting variable is randomly chosen in a [*low_tab_mv*, *high_tab_mv*] interval according to a discrete uniform distribution. Note that pivots that involve the slack variables introduced in constraints (3) do not correspond to moves in the tabu search-sense of the term. These pivots are therefore not part of the computations of the tabu tenures or iteration counts.

Moves are revised simplex pivots (Dantzig and Thapa, 1997), implemented according to the primal partitioning logic of Farvolden, Powell, and Lustig (1992), and executed on the capacitated multicommodity network flow polyhedron defined by $\mathcal{H}$. Since $\mathcal{H}$ corresponds to a design where all arcs are open, the objective function of the corresponding minimum cost network flow formulation reduces to the continuous component of the objective function of the design model (relation (1)):

$$\text{Minimize } z(h) = \sum_{p \in \mathcal{P}} \sum_{l \in \mathcal{L}_i^p} k_l^p h_l^p. \tag{6}$$

This formulation - (6) subject to (2) to (4) - is used to compute dual variables and price non-basic path variables. Then, the value of a potential move is defined as the variation it induces in the value of the objective function of the design formulation (relation (1)). This variation is computed in two steps as the sum of the

1. Variation in the value of (6) due to the pivot;
2. Variation in the total fixed cost of the network, $\sum_{a \in \mathcal{A}} f_a y_a$, due to the corresponding modification (if any) to the vector of design variables y.

The move with the largest decrease is selected and implemented. Non-improving moves (i.e., moves that increase the objective function value) are accepted. Moves conducting to non-feasible solutions are also accepted if they represent the best local option. The corresponding overflows are assigned to artificial arcs with arbitrarily high costs. Note that, using the actual objective function of the design formulation to evaluate moves does imply an extra cost in computing time, but it offers both a precise ordering of potential moves and a direct *aspiration criterion*.

Since not all path variables are available at each iteration, column generation is used to expand the neighborhood and enrich the set of candidates whenever a local optimum solution appears to have been reached. This condition is detected when the best solution identified during the current series of pivot moves, Z_{local}, did not improve for *max_move*

consecutive moves. Two variants of a modified shortest path algorithm over the links of $\mathcal{G}$ are used to generate new paths. In the basic method all arcs may be considered. In the *SAT* variant saturated arcs are eliminated. To capture the interplay between the fixed costs and capacities of the arcs, the reduced arc costs used by the column generation subproblem correspond to a continuous relaxation of the design formulation with the *surrogate* cost function $\overline{c}_a^p = c_a^p + f_a/u_a$, $a \in \mathcal{A}$, $p \in \mathcal{P}$. A fixed number, *k_gen*, of paths is generated for all O–D pairs each time the procedure is executed.

A sequence of neighborhood explorations by pivoting followed by a column generation phase is called a *column generation cycle*. Local search is terminated when no improvement to the current local best solution is obtained after *max_col_gen* consecutive column generation cycles. The method then either stops or proceeds to diversify the search.

The diversification strategy is based on a *discrete* neighborhood and a long term *frequency* memory structure. The diversification neighborhood is defined relative to the design variables and is used to drastically modify the network configuration. Formally, a discrete neighbor $\hat{h}$ of $\bar{h} \in \bar{\mathcal{H}}$ at iteration t is the set of optimal paths in $\cup_{p \in \mathcal{P}} \mathcal{L}_t^p$ that corresponds to the network configuration $\hat{y}$ obtained by "closing" a given number of network arcs in $\bar{y}$. The corresponding *diversification move* is thus a complex sequence of operations which usually includes several (primal and eventually dual) pivots. The long term memory records for how many iterations an arc has been in the basis (i.e., it belonged to at least one basic path). To diversify, one selects a small number of often used arcs and closes them. During the tabu tenure of these arcs, all paths that contain them are not allowed to enter the basis (unless, of course, the aspiration criterion overrides the tabu status). Furthermore, the closed arcs are not available during the column generation phases. The length of the tabu tenure is defined in terms of a number of column generation cycles, *tabu_cycle*, following which the normal termination criterion of the local search procedure is activated.

To initiate the procedure, one may assume that all design arcs are available or that a number of them are closed according to some criterion such as "close those with high fixed cost to capacity ratio up to a given percentage of the total number of arcs". Then, a shortest path is generated for each demand using the surrogate arc costs and demand is sequentially loaded on these paths. The overflow is assigned to the artificial arcs. Starting from this principle, two variants may be used. The first, identified as *YES_initial_solution*, performs simplex pivots without the tabu mechanisms until a first feasible solution is obtained. The second, identified as *NO_initial_solution*, makes use of the local search routine when looking for the first feasible solution, thus initiating the tabu logic from the very start.

The sequential tabu search procedure has been extensively calibrated and tested. This experimental study yielded the parameter settings that are used to define the parallel method presented in this paper.

3 Parallel Strategies

Several classes of strategies may be envisioned when contemplating the development of parallel tabu search methods. See Crainic and Toulouse (1998) for a general review and analysis of parallel metaheuristics, and Crainic, Toulouse, and Gendreau (1997) for a taxonomy of parallel approaches for tabu search. We focus on *multi-search*, also called *multi-thread*, parallel strategies which have proved to offer superior performances (Battiti and Tecchiolli, 1992; Andreatta and Ribeiro, 1994; Rego and Roucairol, 1996; Taillard, 1994; Taillard *et al.*, 1997; Badeau *et al.*, 1997; Gendreau *et al.* 1999; Schulze and Fahle, 1999; Toulouse, Thulasiraman, and Glover, 1999; Ouyang *et al.*, 2000a,b), in particular for network design-type of problems (Crainic, Toulouse, and Gendreau, 1995a,b; Crainic and Toulouse, 1998.

In our implementation of multi-thread parallelism, each process independently executes a complete tabu search procedure as specified in the previous section. The initial solution and particular setting of a number of important search parameters differentiates each tabu search thread from the others. Thus, each thread follows a different *search strategy*. Full implementation details are given in the following section. We implement both an *independent search* method and several variants of a *cooperative search* framework. In the former, all searches proceed independently, and the best solution is collected at the end. In the latter, it is hoped that exchanging information among the tabu search threads will increases the efficiency of the search to yield a higher quality of the final solution.

The cooperation aspect of the parallelization scheme is achieved through asynchronous exchanges of information. Previous studies (Crainic, Toulouse, and Gendreau, 1995a,b) have shown the superiority of this approach over synchronous cooperation. To fully characterize the cooperation process, one has to specify *(i)* the information which is to be shared, *(ii)* when communications may occur, *(iii)* between which threads information is to be exchanged, as well as *(iv)* the utilization each search thread will make of the imported information (Toulouse, Crainic, and Gendreau, 1996).

The information exchanged among search threads has to be meaningful, in the sense that it has to be useful for the decision process of the receiving threads. Information that gives correct indications concerning the current status of the global search or, at least, of some other searches is, in this sense, meaningful. In the implementations described in this paper, threads share information about their respective good solutions identified so far. When a search thread improves its best local search solution Z_{local}, it sends out the value of the solution, $z(h, y_{local})$, as well as its *context* defined as the vector of design variables y_{local}. This scheme is intuitive and simple, and it satisfies the meaningfulness requirement. Note that threads do not exchange full solutions. In particular, neither the flow distributions x, nor the sets of paths h are exchanged. We believe that the efficiency gains thus obtained more than compensate for the fact that the reconstructed solutions

might result in different flow patterns.

Information is shared through a *central memory* or *pool of solutions*. In this scheme, whenever a thread desires to send out information, it sends it to the pool. Similarly, when a thread accesses outside information, it reaches out and takes it from the pool. Communications are initiated exclusively by the individual threads, irrespective of their role as senders or receivers of information. No broadcasting is taking place, and there is no need for complex mechanisms to select the threads that will receive or send information and to control the cooperation. The pool is thus an efficient implementation device that allows for a strict asynchronous mode of exchange, with no pre-determined connection pattern, where no process is interrupted by another for communication purposes, but where any thread may access at all times the data previously sent out by any other search thread.

What information to exchange, when to exchange it, and what to do with the external information once one has it are important issues for the efficiency of cooperative search procedures. Unfortunately, there are not, as yet, well-defined procedures to specify these parameters.

Previous empirical evidence shows, however, that cooperative parallel metaheuristic searches with unrestricted access to shared knowledge may experience serious premature convergence problems. This follows from the excessive bias to the contents of shared memories introduced by intense exchanges of the best solutions found by the search threads and the subsequent stabilization of the shared memory contents and exchanged information. Recent work on formal representation of cooperative processes (Toulouse, 1996; Toulouse, Crainic, and Sansó, 1997, 1998; Toulouse, Crainic, and Thulasiraman, 1998, Toulouse *et al.*, 1998) confirms these observations. It also highlights the fact that simple information exchanges (e.g., best values) may direct the search into unexpected regions, with little concern for an "optimization logic". The implementation of communications and exchanges through a pool of solutions that gives to each search process the control on requesting and accepting external information constitutes a mechanism aimed at addressing these issues. The specification of strategies concerning what external solution to select and when to request it, presented in the next subsections, proceed from the same preoccupations.

We develop five *selection* strategies corresponding to different ways to access the pool data and extract the information requested by an individual search thread. We also examine six *import* criteria to determine when a search tread looks for, and eventually accepts, an external solutions.

3.1 Pool Data Selection Strategies

For all five strategies, the pool stores and manages all the information received from the individual searches and, on request, returns the corresponding solution.

The first strategy (*Sbest*) corresponds to always extracting the solution with the lowest value. Hence, upon request, each individual search thread gets the *global best* solution, (Z^g_{best}, y^g_{best}).

The other variants aim to diversify the information received by the individual searches and go beyond the simple exchange of best solutions strategy. This is in response to a desire to avoid having the threads prematurely converge towards a limited number of regions or solutions. This undesirable phenomenon has been observed in other studies, and it generally emerges when a solution is dominant for a period of time and many (all) threads import it and inflect their search trajectory accordingly. On the other hand, individual searches need good quality information. Hence, while trying to avoid sending always the same solution, one desires to bias the selection process towards solutions with low values. The probabilistic mechanisms described in the following are designed to select with high probability solutions that are good but different from the current global best, while displaying low probabilities of extracting poor (high value) solutions.

Let $\mathcal{S} = \{S_i\}$ be the set of the n solutions that make up the pool at a given time, and assume the set is ordered in increasing order of solution values, i.e. $i < j \Rightarrow z_i < z_j$ for any $S_i, S_j \in \mathcal{S}$. The second strategy (*SProb*) selects among the best solutions based directly on their rank according to the solution values. A probability $P(S_i) = (n+1-i) / \sum_1^n i$ is assigned to each solution S_i. $P(S_i)$ increases as solutions are closer in value to the current global best. The solution returned to a requesting search thread by the *S2* strategy is then randomly selected (Uniform distribution) over $\mathcal{S}$ according to these probabilities.

The third variant (*SMobil*) proceeds of the same principle, but defines the probabilities associated to the solutions in the pool according to their respective *mobility*. Each time a new solution S_{new} is inserted into the ordered pool $\mathcal{S}$, the mobility coefficients m_i of all solutions S_i better than S_{new}, i.e., $z_{new} \leq z_i$, are incremented by 1. Then, at any given point in time, the mobility coefficient of a solution indicates how many worst solutions have been inserted into the pool since the beginning of the computation: the higher the coefficient, the better the solution. (The term *mobility* was coined to reflect the movement towards the top of the list good solutions perform under this strategy.) The probability assigned to each solution S_i then becomes $P(S_i) = (m_i + n - i + 1) / \sum_1^n (m_i + i)$.

Strategies *SHhigh* and *SHlow* represent a first attempt to take into account not only the solution values, but also some measure related to the corresponding design, the so-called *solution context*. The strategies aim to identify common characteristics of good solutions to build an "ideal" individual to which to compare the solutions in the pool.

Selection probabilities are then biased based on these comparisons. The method proceeds as follows. A subset $\mathcal{B} \subseteq \mathcal{S}$ of best solutions (e.g., the best 5% or 10% ones) is first selected. A *design pattern* y^B is then built such that for each design arc $a \in \mathcal{A}$

$$y_a^B = \begin{cases} 1 \text{ if } y_a = 1 \text{ for all } S_i \in \mathcal{B} \\ -1 \text{ if } y_a = 0 \text{ for all } S_i \in \mathcal{B} \\ 0 \text{ otherwise} \end{cases}$$

The Hamming distance between the design pattern and any solution $S_i \in \mathcal{B}$ is then computed as $D_i = \sum_{a \in \mathcal{A}} d_a$, where

$$d_a = \begin{cases} 0 \text{ if } y_a^B = 1 \text{ and } y_a = 1 \\ 0 \text{ if } y_a^B = -1 \text{ and } y_a = 0 \\ 1 \text{ otherwise} \end{cases}$$

and the probability associated to solution S_i is computed as $P(S_i) = (1 - D_i / \sum_1^n D_i)/n$. Variant *SHhigh* then selects among solutions with high probability values, while *SHlow* extracts solutions with low values. This last choice is again justified by a desire to increase the variety of solutions that are returned to individual searches. Here, one selects good solutions according to their objective values ($S_i \in \mathcal{B}$) but which are "far" from the characteristics common to the solutions in the best subset.

3.2 External Solution Import Strategies

To complete the description of the cooperative process, one has to define when individual threads request external information and how the imported solution is used locally. As discussed earlier, the objective of this mechanism is double: to 1) accept external information that may improve the search trajectory while, 2) limit the amount of exchanges such as not to disrupt too much the normal exploration pattern of the particular tabu search thread.

In the sequential tabu search algorithm displayed in Figure 1, three points appear as potentially interesting relative to the import of external solutions:

1. Following the identification of the best current move (Step 6);
2. Before the neighborhood is expanded through a column generation phase (i.e., before Step 10);
3. Before a diversification operation (Step 11).

We exclude from further considerations approaches based on the first option and that interrupt a sequence of pivot moves to access external information. These would generate a very high number of information exchanges, which we want to avoid. Moreover,

by potentially replacing pivot moves with imports of external solutions, such approaches would also transform the tabu search into a random search procedure. We focus, therefore, on strategies based on options 2 and 3. Note that for all strategies, local memories are not touched when the external solution is substituted. Thus, a search thread always proceeds according to the information it gathered locally.

According to the basic strategy (*CDiv*), a solution is requested from the pool before a diversification operation. Then, if the imported solution is better than the current best, the diversification proceeds from the new solution. Otherwise, the imported solution is discarded and the procedure proceeds as usual.

A number of alternative strategies based on option 2 may be envisioned to increase the volume of information exchanges. The alternative criteria thus look for external information once the pivoting sequence stops and before the neighborhood is to be expanded through a column generation phase. If the imported solution is accepted, a new pivoting sequence is initiated. Otherwise, the column generation phase proceeds as usual.

Let y_{out} and Z_{out} be the external solution and its value, respectively. Recall that Z_{local} gives the value of the best solution found during the current sequence of pivot moves, while Z_{prev} denotes the value of the best solution found during the current local search phase up to *the beginning* of the current sequence of pivot moves. We tested the following criteria:

CFdiv: Accept the external solution if better than the local best. This corresponds to a forced diversification, even if the local search trajectory is improving.

CWait: Import of external solution is delayed until after the first diversification; y_{out} is always accepted afterwards (assuming, of course, $Z_{out} < Z_{local}$). It attempts not to disturb the initial local search.

CLoc: Accept y_{out} if "significantly" better than the local best solution:

$$Z_{out} < Z_{local} - \epsilon \text{ with } \epsilon = 10\% \, Z_{out}.$$

It attempts not to interfere with the local search trajectory unless the external solution represents a significant improvement.

CPrev: Accept y_{out} if it represents a more significant improvement relative to y_{local} than y_{local} compared to y_{prev}, the starting point of the current pivoting sequence:

$$Z_{local} - Z_{out} > Z_{prev} - Z_{local}.$$

CRel: Accept y_{out} if better than the current best, relative to the improvement effort of the current pivot move sequence

$$Z_{out} < Z_{local} - \epsilon \text{ with } \epsilon = (Z_{prev} - Z_{local})/nbpiv,$$

where *nbpiv* indicates the number of pivots performed during the last pivoting sequence.

3.3 The Cooperative Tabu Search Method

To summarize:

The *central pool*

- Receives solution values and contexts.
- Manages the set of solutions.
- On request, extracts a solution, according to one of the above-defined strategies (Section 3.1), and returns it to the requesting thread.

The *individual search thread*

- Executes a sequential tabu search procedure (as defined in Section 2) following a specified strategy: initial solution and set of search parameters.
- When it improves Z_{local}, the *local best* solution, sends its value and context to the pool. Solutions are sent during regular local search phases as well as immediately following a diversification move (even if infeasible). Note that sending the improved local best solutions instead of the improved global best increases the size and diversity of the pool.
- Requests a solution from the pool at a moment determined by one of the six import criteria defined previously (Section 3.2). If the external solution is accepted, it becomes the local best, otherwise it is discarded. The tabu search then proceeds normally, using its current memories.

According to the taxonomy of Crainic, Toulouse, and Gendreau (1997), this method can be classified as a *p-C MPDS* strategy: collegial control distributed among p processes, asynchronous communications, multiple initial points, diverse search strategies. The selection of particular selection and import strategies defines and instantiates the cooperation mechanism. Many variants of the basic cooperative parallel algorithm may thus be built. The next section tests and compares several of them.

An *independent search* strategy (identified as *p-RS MPDS* in the same taxonomy) also executes several search threads in parallel, but no communication is allowed among processes, except at the end when the best solution is determined. The same individual search threads are used in our experimentation of cooperative and independent parallel tabu search.

4 Experimentation and Analyses

There are few studies aimed at parallelization strategies for problems that, similarly to the capacitated multicommodity network design formulation, display both a strong combinatorial nature and complex flow structures. A major objective of the experimental phase therefore is to explore the behavior and influence of the various parallelization strategies: independent versus cooperative search, solution pool management, communication interval. The other main objective is to select a parallel strategy and to analyze its performance characteristics for the class of problems at hand.

To analyze the behavior and performance of the parallel tabu search methods, comparisons are made with the output of the sequential tabu search metaheuristic of Crainic, Gendreau, and Farvolden (2000). To further characterize the quality of the solutions, we also refer to the optimal solutions obtained by using the standard branch-and-bound algorithm offered by a widely known commercial software.

The sequential tabu search metaheuristic is programmed in *FORTRAN77*. The cooperation mechanisms are programmed in C. The parallel experiments reported in this section have been performed on a cluster of SUN Ultra Sparc1/140 workstations with a 143 MHz clock, 512 kb of cache memory and 64 Mb of RAM memory (3 out of the 16 computers have 128 Mb of RAM). No other applications were running during the parallel experimentation. The sequential tabu search and branch-and-bound results come from the paper by Crainic, Gendreau, and Farvolden (2000). It is reported in the paper that both procedures were ran on a SUN UltraSparc-II workstation (2 CPUs, but only one allocated to our experiments), with a 296 MHz clock, 2 Mb of cache memory, and 2 Gb of RAM memory.

Section 4.2 examines the comparative performance of the selection and import strategies described previously (Section 3). A particular setting of the cooperative parallel tabu search procedure is then selected and its performance is analyzed in Section 4.3. But first, the description of the general experimental setup.

4.1 Experimentation Setup

To set up the experimental phase, one has to decide how to instantiate each individual tabu search thread. One also has to specify when the global search terminates.

Each tabu search thread is characterized by how the initial solution is obtained and by particular settings of a number of search parameters. For the initial solution procedure, we used three methods: 1) *YES_initial_solution* on the whole network; 2) *NO_initial_solution* on the entire network; 3) *NO_initial_solution* on the network reduced

by closing a number of arcs with high fixed cost to capacity ratios. The search parameters that were varied are: the short term tabu tenure interval ([*low_tab_mv, high_tab_mv*]), the number of consecutive unimproving pivot moves before a column generation is initiated (*max_move*), the number of paths generated (*k_gen*), the proportion of arcs dropped in the initial solution.

When analyzing the performance of a parallel procedure, normally one compares it to the performance of the best implementation of the sequential algorithm for the given problem instance (Barr and Hickman 1993; Barr *et al.* 1995). This is reasonably clear for most numerical procedures and optimization methods. The issue is less simple when parallel metaheuristics are concerned. On the one hand, one certainly does not want to consider the best sequential implementation for each problem instance. Metaheuristics require the calibration of a number of search parameters. Then, normally, a limited set of problems is used to select a set of parameters that perform "well" on these instances. Performance results of the metaheuristic using this "best sequential implementation" are then reported on larger and different sets of problems. It is this metaheuristic version, and the corresponding test problems, that should be used for parallel experiments. On the other hand, a cooperative parallel metaheuristic brings together several procedures. Often, one just takes the best sequential version and starts it from different initial solutions. We do not believe this approach takes full advantage of the cooperation mechanisms. Indeed, some of our earlier experiments have shown that just varying the starting point of the search does not address the issue of building a more robust solution method relative to variations in problem characteristics. Consequently, several different parameter settings are used in our approach to instantiate the individual search threads.

To select the parameter settings, we turn to the experimental results reported by Crainic, Gendreau, and Farvolden (1998, 2000). The calibration of the sequential tabu search yielded a set of preferred parameter settings that were then used to obtain the results reported in that paper. Since we compare to these results, it is only fair that this parameter setting be included as one the cooperative procedures. The same calibration operation has also pointed toward alternative parameter settings, some of which were displaying performances almost as good as those of the preferred set. We used this information to vary the search strategies of the parallel threads. Finally, the sequential experimentation also indicated that for a few very tightly capacitated problem instances, a column generation procedure that explicitly avoids saturated arcs was required. This variant is identified as *SAT*. When included in the parallel experiments, the *SAT* variant always uses the preferred sequential parameter settings.

We designed the parallel experiment such that the "best" (the preferred) sequential parameter settings are always used. The other search strategies combine initial solutions and parameter settings that appeared interesting in sequential testing, but were not exhaustively tested. Therefore, 4-process searches implement the four "best" sequential strategies. 8-process experiments implement on four processors the same settings used in

the 4-process tests, while the other four receive new parameter settings. 16-process experiment use the settings of the 8-process tests, plus eight new ones. In all experiments, the independent and cooperative parallel procedures use exactly the same search strategies for their threads. Note that we use "best" to qualify a given parameter settings in the same sense as Crainic, Gendreau, and Farvolden: a robust set of parameters which offers consistent performance levels over the problems used to calibrate the sequential method. Only ten (10) problems were used to calibrate the sequential tabu search. There is therefore no guarantee that the preferred parameter setting is the "best" in any general sense. It is also noteworthy that neither the sequential nor the parallel tabu search procedures underwent any particular calibration for the parallel experiment. The parameter settings were directly transposed form the sequential experiment without any further study. The only calibration-like experiment that was undertaken concerned the choice of selection and import strategy presented next.

The same problem instances used by Crainic, Gendreau, and Farvolden (2000) are also used in this study. Two sets of problems have been generated. The 196 problems are general transshipment networks, with no parallel arcs and one commodity per origin–destination pair. On each arc, the same unit cost is used for all commodities. Problems differ in the number of nodes, arcs, all of which are design arcs, and commodities. Several instances have been generated for each problem dimension by varying the relative importance of fixed versus variable costs and the capacity of the network compared to the total demand. The problem generators (see [21, 23] for a complete description), as well as the problem instances can be obtained from the authors.

Initial experiments have shown that sending to the pool all solutions that improve the best solution of the current local search phase is counterproductive. Too many very similar solutions are tightly packed together in the pool and the selection criteria no longer discriminate adequately. Thus, the communication strategies have been implemented to avoid this problem. Thus, when an individual search thread performs a series of improving moves, it waits until the series stops and then sends to the pool the last improving solution only. In all experiments, the independent and cooperative parallel procedures are stopped on the same criteria used for the sequential tabu search: three diversification phases. Thus, parallel versus sequential comparisons are easier to perform.

4.2 Communication Strategies and Parallelism

The first phase of experimentation aimed to analyze the influence of a number of parameters on solution quality performance: number of processes, strategy to manage the pool and select a solution following a request, criteria to decide when to import an external solution from the pool. This experimentation was performed on the same set of ten problems used to calibrate the sequential procedure. These problems were selected initially out of a set of 43 problem instances (set C of Section 4.3) to represent the entire

range of network sizes and number of commodities. They also display tight capacities and dominant fixed costs, since the initial sequential experiments had shown that these characteristics make problems more difficult to solve. In tables 1 to 4, the problems are identified by their respective number of nodes, arcs, and commodities. The (F) and (T) letters indicate that capacities are tight and the fixed costs are dominant, respectively.

Tables 1, 2, and 3 display results obtained with four, eight, and sixteen processes (and processors), respectively. For each problem, the tables display results for the independent parallel search (column IND), and the five selection strategies described in Section 3 to extract a solution from the pool: columns *SBest*, *SProb*, *SMobil*, *SHhigh*, and *SHlow*). All experiments use the basic import strategy (*CDiv*) that examines external information before a diversification phase. The performance of each strategy is measured for each problem as the *improvement in solution quality*, in *percentage*, with respect to the corresponding result of the sequential procedure. A negative value indicates that a worst solution has been identified.

A number of interesting observations may be made comparing these results. First, parallel search helps improve the solution quality. Over the one hundred eighty parallel runs, only six found a solution worst than the sequential one, and eleven identify the same solution. In one hundred and sixty-three executions, or over 90% of cases, a better solution is found. At first glance, the improvements might not always seem impressive. They are significant, however, especially when one notices that the sequential results are already very good (the gap of the sequential and parallel procedures with respect to the optimal solution, when known, are indicated in Table 6).

The results also indicate that, as expected, independent search offers interesting results. Using exactly the same search threads, cooperative search outperforms it, however. When four processors are used, the observation is true for two selection strategies; it is almost always true for larger numbers of processors. In fact, the superiority of cooperative search appears to be greater when more processes participate: when eight or sixteen processes are active, all cooperative strategies significantly outperform the independent search approach.

The number of processes clearly has an impact on performance. Not on relatively small problems. And not for the independent search approach, of course. But for most problems, increasing the number of cooperating searches improves the solution quality. On the other hand, increasing the number of processes to sixteen does not contribute, in general, to enhance further the quality of the solutions. In fact, for most strategies, a larger computational effort translates into somewhat lower solution quality. Not all strategies behave in a similar way, however.

On closer examination of the 16-processor results, the solutions sent to the pool by the last eight tabu searches are, for the most part, of lower quality than those of the

PROB (C)	*IND*	*SBest*	*SProb*	*SMobil*	*SHhigh*	*SHlow*
$25, 100, 10, F, T$	0.46	0.46	1.87	1.87	0.61	0.46
$25, 100, 30, F, T$	0	−0.04	0.14	0.14	0.19	−0.17
$100, 400, 10, F, T$	1.50	0.52	1.77	1.77	1.50	1.50
$20, 230, 40, F, T$	0.13	0.02	0	0	0	0.02
$20, 300, 40, F, T$	0.16	0	0.16	0.16	0.16	0
$20, 300, 200, F, T$	5.08	8.01	7.74	7.38	5.50	5.31
$30, 520, 100, F, T$	4.03	2.41	3.13	3.13	3.13	3.13
$30, 520, 400, F, T$	1.37	1.44	1.08	2.12	1.03	1.36
$30, 700, 100, F, T$	1.17	1.65	1.31	1.17	1.17	1.31
$30, 700, 400, F, T$	2.77	2.00	1.70	1.70	0.55	0.54
Av. Improvement	1.67	1.64	1.89	1.95	1.38	1.38

Table 1: Selection Strategies, 4 Processors: Improvement (in%) Over Sequential Search

PROB (C)	IND	*SBest*	*SProb*	*SMobil*	*SHhigh*	*SHlow*
$25, 100, 10, F, T$	0.46	0.46	1.19	0.81	0.61	0.46
$25, 100, 30, F, T$	0	−0.04	0.19	0.19	0.19	−0.17
$100, 400, 10, F, T$	1.50	0.52	−0.84	−0.46	1.52	3.16
$20, 230, 40, F, T$	0.13	0.02	0.02	0.02	0	0.13
$20, 300, 40, F, T$	0.16	0	0.16	0.16	0.16	0.16
$20, 300, 200, F, T$	5.08	9.26	8.84	5.93	7.98	8.94
$30, 520, 100, F, T$	4.03	3.27	4.03	3.51	4.03	3.64
$30, 520, 400, F, T$	1.37	3.68	5.16	4.20	3.60	3.25
$30, 700, 100, F, T$	1.17	1.72	1.71	1.71	1.53	1.31
$30, 700, 400, F, T$	2.77	3.70	4.06	2.77	2.77	3.53
Av. Improvement	1.67	2.26	2.45	1.88	2.24	2.24

Table 2: Selection Strategies, 8 Processors: Improvement (in%) Over Sequential Search

PROB (C)	IND	*SBest*	*SProb*	*SMobil*	*SHhigh*	*SHlow*
$25, 100, 10, F, T$	0.56	0.61	0.79	0.46	0.50	0.79
$25, 100, 30, F, T$	0	0.09	0.11	0.14	0.23	0.23
$100, 400, 10, F, T$	1.50	1.89	1.50	1.50	1.77	1.77
$20, 230, 40, F, T$	0.13	0	0.02	0.02	0.02	0.02
$20, 300, 40, F, T$	0.16	0.16	0.16	0.16	0.16	0.16
$20, 300, 200, F, T$	5.08	7.98	7.72	7.98	7.54	8.05
$30, 520, 100, F, T$	4.03	3.51	4.03	4.03	3.51	3.51
$30, 520, 400, F, T$	1.37	1.93	1.37	2.08	3.90	1.42
$30, 700, 100, F, T$	1.17	1.50	1.17	1.17	1.33	1.68
$30, 700, 400, F, T$	2.77	3.47	2.97	2.84	2.84	2.77
Av. Improvement	1.67	2.12	1.98	2.04	2.18	2.18

Table 3: Selection Strategies, 16 Processors: Improvement (in%) Over Sequential Search

other search threads. And few make it to the top of the pool list. However, their presence and the modifications it forces on the ordering of the solutions in the pool, appears to significantly disturb the search patterns of most threads under most selection strategies, in particular strategies *SBest* and *SProb* that rely directly on the solution values. Strategies *SHhigh* and *SHlow* fare better because their evaluations are based not on direct comparison of solution values but on distances from a constructed partial solution that reflects the common attributes of a group of best solutions. Thus, the impact of any particular solution is less important than for the other two strategies. However, even for the *SHhigh* and *SHlow* selection strategies the contributions of the last eight processes to the pool do not appear to be of sufficient quality to compensate for the increased volatility of the pool. Only for *SMobil* a larger number of processors has a beneficial impact on solution quality. A larger number of lower quality solutions reinforces the positions of the solutions that top the pool, thus reinforcing the global search pattern induced by the strategy when only eight processors are available. But even for *SMobil*, the improvement is marginal and does not compensate for the additional computational effort. Thus, for the current set of problems and parameter settings, eight processors appear appropriate.

Over all trials, the traditional approach of always exchanging the best solutions is often competitive and generally outperforms the independent search strategy. It is not the best, however. Strategies that increase the diversity of the solutions returned to requesting processes do as well and often better. Strategies that bias the selection probabilities by taking into account the relative position of solutions in the pool, *SProb* and *SMobil*, fare generally well. In our trials, the two strategies behave rather similarly, *SMobil* being less disturbed when the size of the pool is increased by a significant number of solutions of lower quality.

Strategies based on building a partial solution that reflects characteristics common to a set of best solutions and using the distance between a solution and this image to bias the selection probability display interesting performances as well. Certainly not appropriate when few processors are used, strategies *SHhigh* and *SHlow* are competitive when this number is increased. This follows from the fact that when few search threads are used, all good solutions available in the pool are sent by the same processes. The good solutions are in fact quite strongly interrelated, the better ones often resulting from a search process started at some other solution in the candidate set. The similarity of characteristics is thus not meaningful. A larger number of search threads makes for a more diverse set of good solutions in the pool and their common characteristics correspond to meaningful information. Although interesting and encouraging, we believe that more work is needed to refine this concept.

Recall that the *SAT* variant of the tabu search procedure forbids saturated arcs to be considered during the column generation process. This modification was introduced to tackle particularly tightly capacitated problem instances. Including into the cooperation

PROB (C)	IND	*SBest*	*SProb*	*SMobil*	*SMobil-*	*SHhigh*	*SHlow*
$25, 100, 10, F, T$	0.46	0.46	0.50	0.50	0.65	0.50	0.50
$25, 100, 30, F, T$	0.22	−0.04	0.22	0.22	0.39	0.22	0.22
$100, 400, 10, F, T$	1.50	0.88	1.52	2.63	0.66	1.52	−0.84
$20, 230, 40, F, T$	0	0.02	0	0.02	0.02	0	0
$20, 300, 40, F, T$	0.26	0	0.26	0.26	0.10	0.26	0.26
$20, 300, 200, F, T$	5.50	9.75	7.36	9.36	5.82	7.36	5.50
$30, 520, 100, F, T$	4.03	4.54	3.33	2.32	2.71	3.23	4.03
$30, 520, 400, F, T$	0.54	2.95	2.66	3.14	3.25	1.92	0.54
$30, 700, 100, F, T$	1.17	2.00	1.37	1.83	2.20	1.86	1.17
$30, 700, 400, F, T$	1.73	3.67	3.97	3.95	3.31	4.32	1.73
Av. Improvement	1.54	2.42	2.12	2.42	1.91	2.12	1.31

Table 4: Selection Strategies with *SAT*, 8 Processors : Improvement (in%) Over Seq.

at least one search thread that implements the *SAT* strategy should thus increase the robustness of the parallel method relative to the characteristics of the problem instances. Experimental results with eight search threads, one of which implements *SAT* (and the preferred sequential parameter setting), are displayed in Table 4. With the exception of *SHlow*, all cooperative strategies perform very well and compare advantageously to the non-*SAT* variants. Given its overall performances the *SMobil* selection strategy with the *SAT* variant has been chosen for the experiments described in Section 4.3.

All the preceding results, as well as those discussed later on (see Section 4.3), are obtained by using the basic communication criterion *CDiv*: a search thread requests an external solution just before a diversification move. This approach clearly outperformed the five others. Table 5 displays statistics to support this allegation. The figures represent the average improvement in solution quality achieved by the parallel methods as compared to the sequential tabu search. Each parallel method is defined by a strategy to extract solutions from the pool and by a communication criterion. The improvements are averaged over the same ten problems. Eight processors were used, with no *SAT* process. The only other criterion that displays improvements over the sequential approach is *CWait* where acceptance of external solutions is delayed after the first diversification move, which is also the moment when one starts requesting external solutions according to the *CDiv* strategy. This emphasizes the importance of an undisturbed initial search phase. The results also clearly support the idea that too frequent communications are not beneficial.

An interesting question in cooperative search is just how much the individual threads contribute to the common goal. The results already reported in this section show that cooperation is beneficial and that it improves over independent search. They also illustrate, however, that cooperation has to be carefully defined and implemented. We have

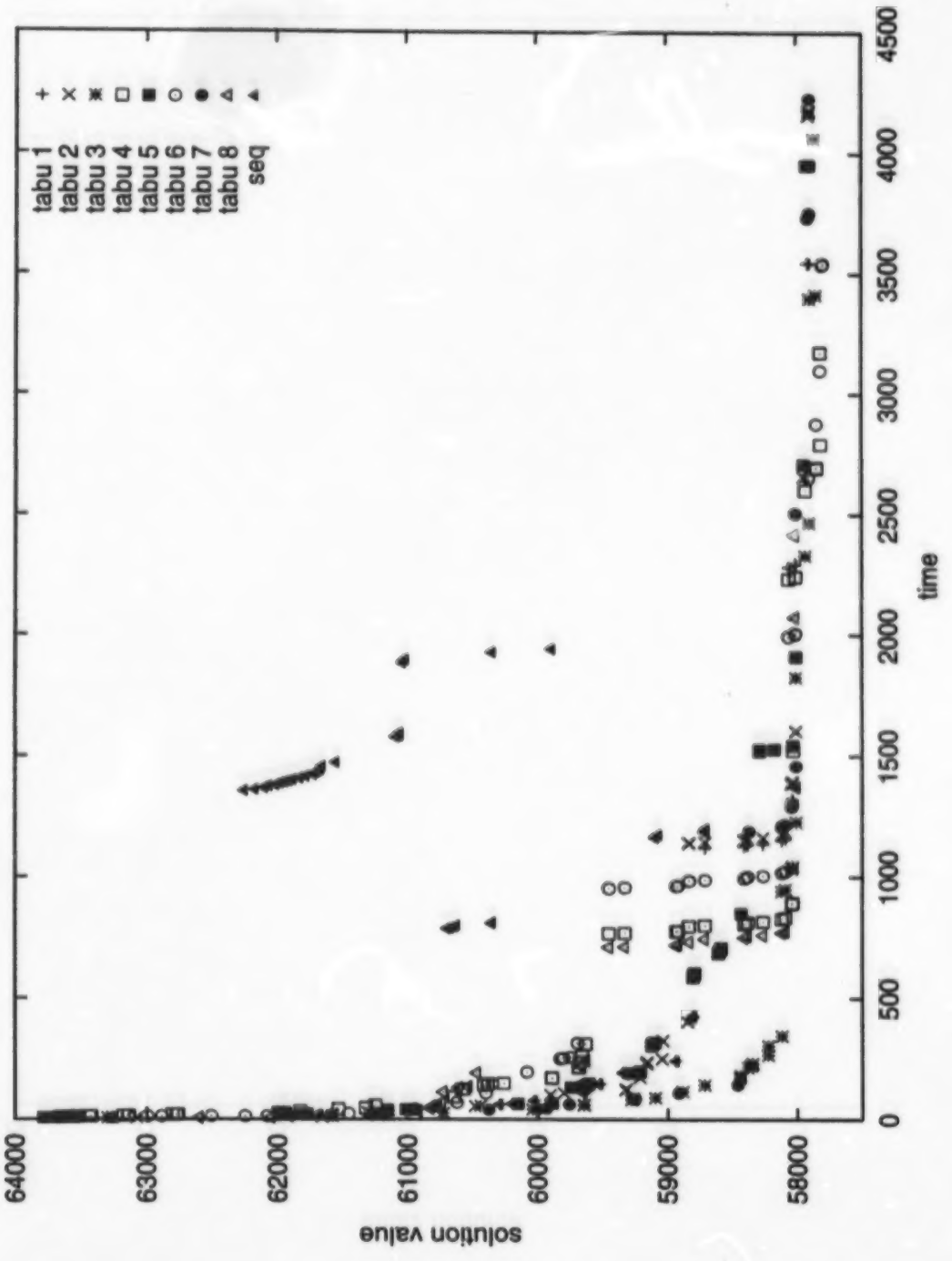

Figure 2: Evolution of the Pool: Prob. 30,700,100,F,T, Strategy *SBest*, 8 Processors

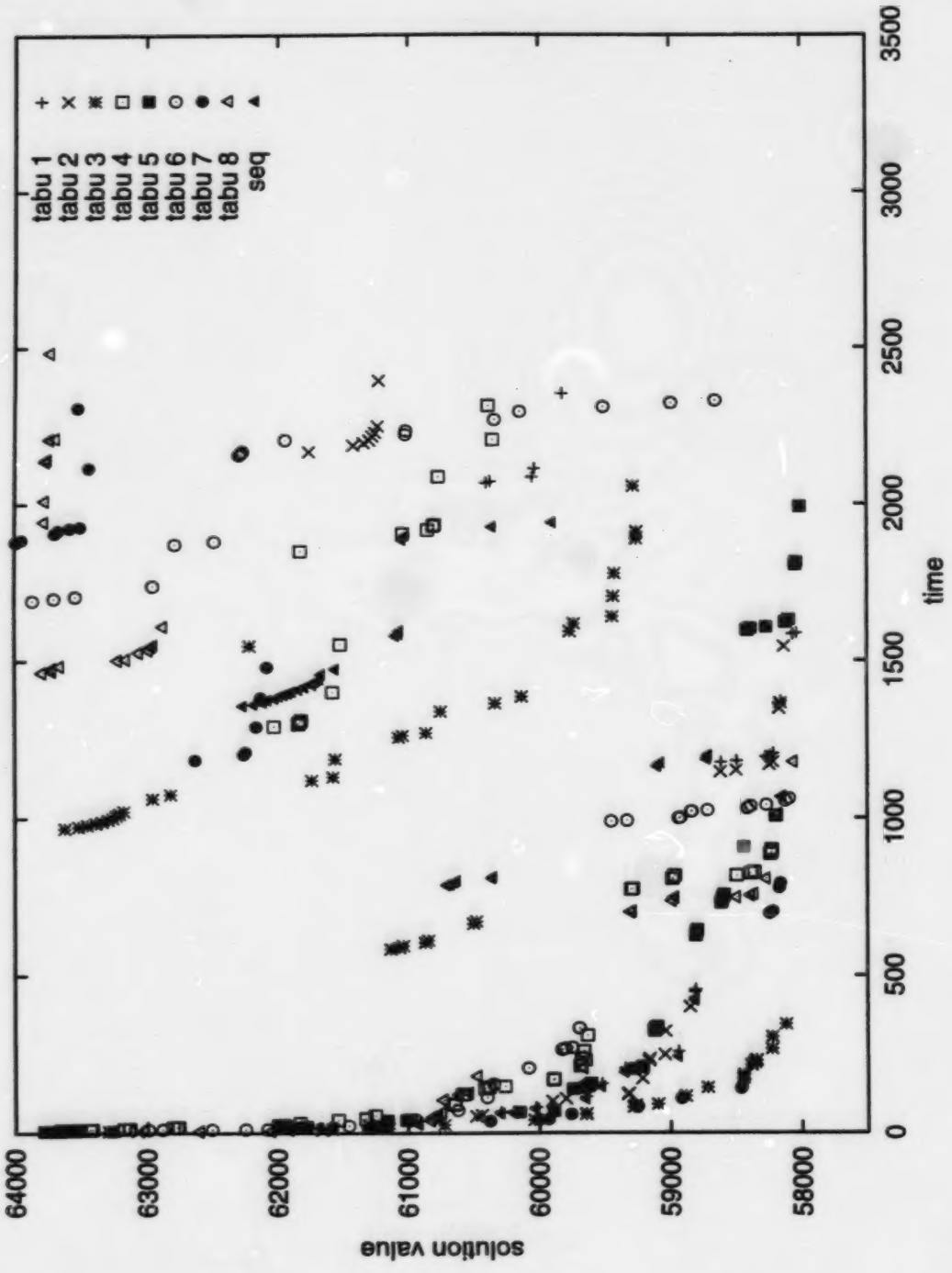

Figure 3: Evolution of the Pool: Prob. 30,700,100,F,T, Strategy *SMobil*, 8 Processors

Pool Strategies	Communication Criteria					
	CDiv	*CFdiv*	*CWait*	*CLoc*	*CPrev*	*CRel*
SBest	1.69	-0.37	0.64	-0.67	-0.01	-0.79
SProb	1.88	-0.25	0.50	-0.71	0.43	-0.68
SMobil	1.30	0	0.82	-0.65	0.26	-0.45
SHigh	1.67	-0.18	0.45	-0.88	-0.08	-0.52
Average	1.64	-0.20	0.60	-0.73	0.15	-0.61

Table 5: Communication Criteria on 8 Processors - Average Improvement

monitored the evolution of the pool for many problems solved using a large number of the parallel methods presented herein. We observed that all processes contributed to the pool, almost all contributing the best solution at one moment, especially when four or eight processes cooperate. Figures 2 and 3 illustrate the evolution of the pool when strategies *SBest* and *SMobil*, respectively, are used to solve problem 30,700,100,F,T on eight processors. The values of the solutions are plotted at the moment they enter the pool. A particular symbol is used for each search thread. The evolution of the sequential tabu search procedure is also indicated. The ▲ symbol is used to plot the values of the solutions found by the sequential procedure on one of the computers used for the parallel runs.

When the best solution in the pool is always returned to a requesting thread (Figure 2), one notices that the values of the solutions transmitted to the pool tend to cluster together. Even here cooperation is beneficial, however: the best solution significantly improves over the sequential one and is identified by a process not in the four "best" (process 6, symbol ⊙). When, on the other hand, the *SMobil* selection strategy is used, the solutions returned to requesting threads are more diverse. Then, as illustrated in Figure 3, the search trajectories of the individual threads are modified (the series of symbols that seem to descend from the top of the figure passed the initial descent, indicate that the corresponding threads have significantly diversified the search following the import of an external solution) and the solutions in the pool are also more diverse, at least in their corresponding solution values. Thread 5 (symbol ■) identifies the best solution in this case.

The graphs for other strategies and problems display similar behaviors. This indicates that 1) the extraction strategies we propose achieve their goal of a more diversified global search trajectory and, 2) external solutions are indeed accepted. Imported external solutions then modify the search path of the thread and, eventually, allow to identify better solutions.

To close this part of the result analysis, we briefly touch upon an intriguing issue related to the instantiation of cooperative parallel procedures: the impact of the "quality" of the individual search strategies included in a particular experiment. We have indicated

earlier in this section that our choices were dictated by the sequential experiments. We believe such a policy to be very reasonable, since one always have some information concerning the behavior of the procedure in a sequential setting. On the other hand, it is not at all clear how else to select search strategies, especially if one 'desires to avoid extensively calibrating the parallel method. Consequently, we performed a very limited experiment. We discarded the two "best" sequential parameter settings and, for eight searches, we selected the next eight strategies (i.e., from the third to the tenth). We then run the ten test problems, with the *SMobil* selection strategy and the *SAT* variant. The results are displayed in column *SMobil-* of Table 4.

The results are very good. Certainly competitive with the experiment that included the two best parameter settings. And better than Independent search. This provides no indication, however, on how to choose search strategies for cooperative parallel implementations. We still believe that basing this selection process on the sequential results is a reasonable approach. In the same time, however, it is an indication of the intrinsic value of parallel computation and cooperation.

4.3 Performance Analysis

Following these experiments, we solved the full sets of test problems using eight processors, the basic communication criterion *CDiv*, the *SMobil* selection strategy, and one of the eight processes running the *SAT* version of the sequential tabu search.

The first set is made up of 43 problem instances. These problems are identified with the letter C and a quintuplet which indicates (i) the number of nodes; (ii) the number of arcs; (iii) the number of commodities; (iv) if the fixed costs are relatively high (F) or low (V) compared to the variable costs; and (v) if the problem is tightly (T) or somewhat loosely (L) capacitated. This set includes the ten problems used to calibrate the sequential tabu search (Crainic, Gendreau, and Farvolden 2000) and to select the communication strategies. It also includes the largest problem instances solved, up to 700 design arcs and 400 commodities.

Table 6 displays computational results for problems in set C. For comparison purposes, column *SEQ OPT GAP* displays the *optimality gap* between the solution obtained by the sequential tabu search and the corresponding "optimal" solution. The gap is computed with respect to the optimal solution and is displayed as a percentage. The optimal solution is obtained by using the branch-and-bound algorithm of CPLEX version 4.0, with primal simplex-based bounding. The branch-and-bound algorithm has been run for six hours (21000+ CPU seconds) but it still had difficulties solving the larger problems (actually, almost all problems with 100 or more commodities). The problems for which this time limit has been attained are indicated with a *(t)*. When the branch-and-bound has identified a feasible solution, the "optimality" gap is then computed with respect to

the best solution found: it is the best available, even if it may be far from the actual optimum. The time required by the sequential tabu search appears in CPU seconds in column *SEQ TIME*. These results are from Crainic, Gendreau, and Farvolden (2000).

Column *PTS OPT GAP* displays the optimality gap of the parallel procedure with respect to the same branch-and-bound solution used to for the sequential procedure gap. When the branch-and-bound did not identify any feasible solution (signaled in Table 6 by a $-$ symbol in column *SEQ OPT GAP*), the percentage of improvement of the parallel solution relative to the sequential one is displayed instead. Column *PTS TIME* displays the CPU seconds required by the parallel procedure to complete the same number of diversification phases (3) as the sequential tabu search. Column *BEST PTS TIME* indicates the CPU seconds required by the parallel procedure to find the best overall solution (used to compute the optimality gap). Recall that the parallel experiments have been performed on computers slower than those used for the sequential runs.

The second set of problems, identified with the letter **R**, comprises 153 instances. To generate these problems, the numbers of nodes, design arcs, and commodities are systematically varied, and for each combination of dimensions, nine problem instances are generated to account for various levels of variable to fixed cost and total demand to total capacity ratios. In this way, one may examine to the impact on performance of problem of modifications to one problem characteristic only. The network dimensions appear in Table 7. Three levels of *fixed cost* ratio have been used: *F01* (Fixed cost ratio = 0.01, *F05* (Fixed cost ratio = 0.05), and *F10* (Fixed cost ratio = 0.10). A high fixed cost ratio indicates that fixed costs are relatively high compared to the total transportation cost of the problem instance and, in most cases, makes the problem harder to solve. Three levels have also been used for the *capacity ratio*: *C1* (Capacity ratio = 1.0), *C2* (Capacity ratio = 2.0), and *C3* (Capacity ratio = 8.0). A high capacity ratio indicates that the total capacity of the network is relatively low compared to the total commodity demand. The tighter the capacity, the more difficult the problem instance is to solve. Thus, the "easiest" problems generally have F01 and C1 as fixed cost and capacity ratios, respectively, while instances with F10 and C8 ratios are more difficult.

The large amount of data precludes the inclusion of detailed results for problems **R** (these may be found in Crainic and Gendreau, 1998). Tables 7 and 8 present global performance statistics over the 153 instances. Table 7 displays the average optimality gaps for the sequential and parallel procedures according to the problem dimensions. Table 8 presents the same information but distributed according to the combination of fixed cost and capacity ratios. The figures between parentheses that follow some average gaps indicate the number of problem instances in that particular class that were solved to optimality.

The results are very satisfactory. With very few exceptions (2 C-instances and 4 in set **R**), the parallel procedure significantly reduces the optimality gap, dramatically so

PROB		SEQ OPT GAP	SEQ TIME	PTS OPT GAP	PTS TIME	BEST PTS TIME
20,230, 40,V,L		0.28	71.29	0.16	216.26	79.80
20,230, 40,F,T		0.09	90.28	0.09	218.37	10.28
20,230, 40,F,T		0.17	121.79	0.12	235.23	93.02
20,230,200,V,L	(t)	29.38	504.50	12.67	4471.12	2479.90
20,230,200,F,L	(t)	34.82	491.63	27.85	4182.90	2003.79
20,230,200,V,T	(t)	20.40	548.36	14.80	3587.83	1680.98
20,230,200,F,T	(t)	32.68	889.69	21.32	1510.57	850.99
20,300, 40,V,L		0.12	71.05	0.12	230.50	21.63
20,300, 40,F,L		0.53	113.44	0.06	301.08	43.93
20,300, 40,V,T		0	145.33	0	299.21	63.42
20,300, 40,F,T		0.35	123.42	0.10	283.03	77.04
20,300,200,V,L	(t)	17.15	982.21	9.44	6281.34	2331.23
20,300,200,F,L	(t)	29.54	1316.75	15.85	5961.83	3260.63
20,300,200,V,T	(t)	10.17	938.29	11.09	4736.64	823.11
20,300,200,F,T	(t)	20.37	1065.88	9.23	6222.3	2763.56
25,100, 10,V,L		0	5.60	0	20.91	1.25
25,100, 10,F,L		6.35	8.37	5.27	32.95	12.24
25,100, 10,F,T		3.52	17.10	3.00	43.53	13.89
25,100, 30,V,T		0	16.57	0	167.91	20.73
25,100, 30,F,L		4.72	33.01	7.04	82.39	39.94
25,100, 30,F,T		1.07	71.84	0.85	131.09	49.23
30,520,100,V,L	(t)	4.29	995.64	1.94	2629.52	1278.35
30,520,100,F,L	(t)	7.94	939.24	4.86	3069.79	2150.41
30,520,100,V,T	(t)	1.97	1218.52	1.21	3225.35	644.06
30,520,100,F,T	(t)	8.03	670.29	3.89	5246.94	3960.93
30,520,400,V,L	(t)	11.33	5789.27	8.63	11337.30	4536.26
30,520,400,F,L	(t)	–	6406.62	5.22	29132.10	10154.20
30,520,400,V,T	(t)	–	6522.23	3.34	19754.50	12733.80
30,520,400,F,T	(t)	–	8415.24	2.52	19167.80	18841.70
30,700,100,V,L		2.90	1265.11	1.23	3192.03	1766.84
30,700,100,F,L	(t)	7.92	1479.59	6.68	7029.04	2371.26
30,700,100,V,T	(t)	1.12	2426.02	1.12	6176.90	896.02
30,700,100,F,T	(t)	5.56	1735.72	3.19	5693.10	2175.09
30,700,400,V,L	(t)	–	12636.20	1.08	18445.50	10688.60
30,700,400,F,L	(t)	–	11367.70	8.05	32752.70	25556.90
30,700,400,V,T	(t)	–	15879.50	1.84	19778.70	6504.07
30,700,400,F,T	(t)	–	11660.40	3.96	29948.90	19298.50
100,400, 10,V,L		0.18	32.66	0.04	596.56	2.57
100,400, 10,F,L	(t)	−1.00	33.00	−1.64	469.31	174.81
100,400, 10,F,T	(t)	−0.44	81.23	−1.95	617.74	189.81
100,400, 30,V,T		0	277.50	0	800.34	365.80
100,400, 30,F,L	(t)	12.23	100.16	8.83	1438.78	492.50
100,400, 30,F,T	(t)	2.89	215.71	1.37	922.07	497.45

Table 6: Performance Results - C Problems

$\|\mathcal{P}\|, \|\mathcal{N}\|, \|\mathcal{A}\|$	*SEQ*	*PTS*	$\|\mathcal{P}\|, \|\mathcal{N}\|, \|\mathcal{A}\|$	*SEQ*	*PTS*	$\|\mathcal{P}\|, \|\mathcal{N}\|, \|\mathcal{A}\|$	*SEQ*	*PTS*
10,10,25	1.85	1.97	10,10,50	2.33	1.83	10,10.75	3.67	1.94
25,10,25	0.85	0.86	25,10,50	2.78	1.33	25,10,75	3.13	1.88
50,10,25	0.54	0.44	50,10,50	7.45	5.51	50,10,75	8.94	3.93
40,20,100	5.00	3.36	40,20,200	8.43	5.86	40,20,300	6.47	6.08
100,20,100	9.03	4.93	100,20,200	16.87	9.67	100,20,300	20.81	14.25
200,20,100	8.07	4.12	200,20,200	24.41	14.74	200,20,300	23.25	13.50

Table 7: Average Optimality Gaps (in %), Problems **R**, by Network Dimensions

	C1		*C2*		*C8*	
	SEQ	*PTS*	*SEQ*	*PTS*	*SEQ*	*PTS*
F01	3.44 (5)	2.48 (6)	3.06 (5)	1.67 (5)	3.17 (1)	1.95 (2)
F05	13.85 (2)	7.85 (2)	9.69 (1)	6.76 (2)	6.33	6.41
F10	22.34 (2)	12.09 (2)	13.92	8.99 (2)	8.05	4.94

Table 8: Average Opt. Gaps (in%), Problems **R**, by Fixed Cost and Capacity Ratios

for most cases. This behaviour is consistent over the entire range of problem dimensions, number of commodities, and complexity (in terms of importance of fixed costs and capacity tightness). This, for us, constitutes a clear indication of the robustness of the cooperative parallel method.

Table 6 displays the CPU times required by the sequential and parallel procedures. Even accounting for the fact that the sequential procedure was run on a computer faster than the machines used for the parallel experimentation, it would appear that the parallelization requires longer computing times. The same observations may be made relative to the resolution of problems **R** (Crainic and Gendreau, 1998). This is not surprising, however, given the setup of the experiments reported in this paper.

Recall that the parallel procedure is stopped on the same criterion as the sequential one: after a number of diversification phases (three for the experiments reported in this paper). Consequently, *a priori*, there cannot be a gain in computing time. Given dedicated computers, the parallel procedure with p processors should take about the same wall clock time as the sequential method, for a total work effort p times superior. The pool management activities do not significantly impact the time of the parallel procedure. The termination operations do have a slight impact, since one has to make sure all processes have completed their search. Communications have an impact on the time of each process. At each diversification point, a process has to stop, get an external solution, and decide on its next step. These operations do not explain, however, all the differences in computing times.

Two additional factors explain these differences. First, continuing the search from an external solution requires a more significant set up time than a normal diversification. Recall that only the context of the solution, the y variables, are exchanged. No information is exchanged concerning the path flows or the simplex basis. Consequently, when a completely foreign design vector is imported, the work required to reconstruct a feasible solution (and that includes generating new paths) may be more significant than recovering from the closure of a few arcs. This time could be reduced either by changing the current dual simplex procedure we are using (one could solve the multicommodity flow problem from scratch, for example), or by exchanging more information. One could, for example, store the full bases that correspond to the solutions in the pool. Another possibility is to build an associated pool of good paths, that is, paths used in the solutions sent to the pool, out of which individual threads could extract paths as needed.

The second factor has to do with the exploration each method performs. An imported solution often sends the search in a direction different from that of a internal diversification move. The new region is then more different from the ones already explored than is the case during a normal sequential search. As a consequence, the following local search phases are longer. Thus, cooperation makes for a more thorough exploration of the solution space compared to the sequential search. This, obviously, requires more time.

The question than appears to be: For similar solution quality, is there any gain in computing time using parallel implementations? For independent search, the correct answer is *No*. This follows directly from the design of the method: control may be exerted only once all threads have completed their search. Therefore, the total wall clock CPU time of independent search equals the CPU time of the slower search process plus the time required to collect the individual solutions and to select the best. (When, in the literature, one encounters claims of speed ups obtained using independent search, inevitably the method is compared to multi-start sequential heuristics. But then, in reality, one compares the same parallel method implemented on two different computing environments.)

For cooperative search, the answer is *Yes*. Several observations point towards this conclusion. First, the column *BEST PTS TIME* in Table 6 displays the moment at which the best solution of the parallel tabu search has been first encountered. This time is generally significantly smaller than the total execution time. Second, the analysis of the evolution of the best solution value among the solutions in the pool reveals that the cooperative parallel tabu search achieves very good quality solutions rapidly, significantly faster than the sequential procedure.

This is clearly illustrated in Figures 2 and 3 where each solution included in the pool is represented by a symbol indicating the thread that produced it with coordinates (*time of entry in pool, solution value*). The solution values obtained by the sequential procedure

are represented as well (the ▲ symbol). To facilitate the comparisons, the sequential and parallel strategies have been run on the same computers. It is then clear that compared to the sequential procedure 1) the same quality of solution has been obtained by the parallel procedure in significantly less computing time and, 2) many solutions of better quality are found much more rapidly. These observations hold for over 90% of the problems we solved and, in fact, the potential time savings appear more important when problems become larger in terms of number of design arcs and, especially, number of commodities. Significant speedups could thus possibly be achieved using cooperative tabu search.

To take advantage of this property, one has to stop the parallel search sooner: not on a number of iterations performed by each individual thread, but rather on a measure related to the value of the best solution value and its rate of improvement. The general idea is to stop the search as soon as the rate of improvement of the best solution "levels off". We have briefly examined a few stopping criteria based on the evolution of the best solution in the pool. The results are encouraging. They also indicate, however, that stopping criteria might be quite difficult to control and calibrate. Moreover, the impact of such stopping criteria on the sequential method has to be correctly evaluated as well. Evidently, more research is required in this area.

5 Conclusions

We have presented a parallel cooperative tabu search method for the fixed charge, capacitated, multicommodity network design problem. The asynchronous parallel method makes use of a pool of good solutions gradually built by the individual search threads. It is through this device that individual threads have access to information sent out by the other threads and communications are established. Various communication policies, as well as several strategies to handle the exchanged information, have been introduced, analyzed, and compared. The resulting parallel procedure displays excellent performances in terms of solution quality and solution times.

The experiments we conducted show that parallel implementations find better solutions than sequential ones. We have also showed that, when properly designed and implemented, cooperative search outperforms independent search strategies. We believe this result to be true for problems where each configuration of the solution space corresponds to a complex problem, which constrains the number of total moves each individual search thread will execute.

Among the many interesting research subjects that follow from this work, we would like to point to the need to further study the behavior and properties of parallel cooperative tabu search. Issues related to global memories and the global guidance of the search (towards intensification and diversification moves in the space of the entire paral-

lel search, for example), are of particular interest. Of course, these issues are of equally relevance and importance to all parallel metaheuristics and hybrids.

Acknowledgments

We thank Nathalie Talbot, Alexandre Le Bouthiller, and Dominique Tourillon who performed the implementations and the innumerable tests required by this project. We would like to thank also two anonymous referees. Their comments have helped us produce a better paper.

Funding for this project has been provided by the Natural Sciences and Engineering Council of Canada, through its Research and Strategic Grant programs, and by the Fonds F.C.A.R. of the Province of Québec.

References

[1] Andreatta, A.A. and Ribeiro C.C. A Graph Partitioning Heuristic for the Parallel Pseudo-Exhaustive Logical Test of VLSI Combinational Circuits. *Annals of Operations Research*, 50:1–36, 1994.

[2] Badeau, P., Guertin, F., Gendreau, M., Potvin, J.-Y., and Taillard, É.D. A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows. *Transportation Research C: Emerging Technologies*, 5(2):109–122, 1997.

[3] Balakrishnan, A., Magnanti, T.L., and Mirchandani, P. Network Design. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 311–334. John Wiley & Sons, New York, NY, 1997.

[4] Balakrishnan, A., Magnanti, T.L., and Wong, R.T. A Dual–Ascent Procedure for Large–Scale Uncapacitated Network Design. *Operations Research*, 37(5):716–740, 1989.

[5] Barr, R.S., Golden, B.L., Kelly, J.P., Resende, M.G.C., and Stewart, W.R. Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics*, 1(1):9–32, 1995.

[6] Barr, R.S. and Hickman, B.L. Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts Opinions. *ORSA Journal on Computing*, 5(1):2–18, 1993.

[7] Battiti, R. and Tecchiolli, G. Parallel Based Search for Combinatorial Optimization: Genetic Algorithms and TABU. *Microprocessors and Microsystems*, 16(7):351–367, 1992.

[8] Crainic, T.G. Long Haul Freight Transportation. In Hall, R.W., editor, *Handbook of Transportation Science*, pages 433–491. Kluwer, Norwell, MA, 1999.

[9] Crainic, T.G. Network Design in Freight Transportation. *European Journal of Operational Research*, 122(2):272–288, 2000.

[10] Crainic, T.G., Frangioni, A., and Gendron, B. Bundle-Based Relaxation Methods for Multicommodity Capacitated Network Design. *Discrete Applied Mathematics*, 2000. forthcoming.

[11] Crainic, T.G. and Gendreau, M. Cooperative Parallel Tabu Search for Capacitated Network Design. Publication CRT-98-71, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 1998.

[12] Crainic, T.G., Gendreau, M., and Farvolden, J.M. A Simplex-Based Tabu Search Method for Capacitated Network Design. Publication CRT-98-37, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 1998.

[13] Crainic, T.G., Gendreau, M., and Farvolden, J.M. A Simplex-Based Tabu Search Method for Capacitated Network Design. *INFORMS Journal on Computing*, 12(3):223–236, 2000.

[14] Crainic, T.G. and Toulouse, M. Parallel Metaheuristics. In T.G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 205–251. Kluwer, Norwell, MA, 1998.

[15] Crainic, T.G., Toulouse, M., and Gendreau, M. Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements. *OR Spektrum*, 17(2/3):113–123, 1995a.

[16] Crainic, T.G., Toulouse, M., and Gendreau, M. Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements. *Annals of Operations Research*, 63:277–299, 1995b.

[17] Crainic, T.G., Toulouse, M., and Gendreau, M. Towards a Taxonomy of Parallel Tabu Search Algorithms. *INFORMS Journal on Computing*, 9(1):61–72, 1997.

[18] Dantzig, G.B. and Thapa M.N. *Linear Programming: Introduction*. Springer-Verlag, Berlin., 1997.

[19] Farvolden, J.M., Powell, W.B., and Lustig, I.J. A Primal Partitioning Solution for the Arc–Chain Formulation of a Multicommodity Network Flow Problem. *Operations Research*, 41(4):669–694, 1992.

[20] Gendreau, M., Guertin, F., Potvin, J.-Y., and Taillard, É.D. Tabu Search for Real-Time Vehicle Routing and Dispatching. *Transportation Science*, 33(4):381–390, 1999.

[21] Gendron, B. and Crainic, T.G. Relaxations for Multicommodity Network Design Problems. Publication CRT-965, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 1994.

[22] Gendron, B. and Crainic, T.G. Parallel Branch-and-Bound Algorithms: Survey and Synthesis. *Operations Research*, 42(6):1042–1066, 1994a.

[23] Gendron, B. and Crainic, T.G. Bounding Procedures for Multicommodity Capacitated Network Design Problems. Publication CRT-96-06, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 1996.

[24] Gendron, B., Crainic, T.G., and Frangioni, A. Multicommodity Capacitated Network Design. In B. Sansó and P. Soriano, editors, *Telecommunications Network Planning*, pages 1–19. Kluwer, Norwell, MA, 1998.

[25] Glover, F. Tabu Search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[26] Glover, F. Tabu Search – Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[27] Glover, F. and Laguna, M. *Tabu Search*. Kluwer, Norwell, MA, 1997.

[28] A. Grama and V. Kumar. Parallel Search Algorithms for Discrete Optimization Problems. *ORSA Journal on Computing*, 7(4):365–385, 1995.

[29] Magnanti, T.L. and Wong, R.T. Network Design and Transportation Planning: Models and Algorithms. *Transportation Science*, 18(1):1–55, 1986.

[30] Minoux, M. Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications. *Networks*, 19:313–360, 1986.

[31] Ouyang, M., Toulouse, M., Thulasiraman, K., Glover, F., and Deogun, J.S. Multi-Level Cooperative Search: Application to the Netlist/Hypergraph Partitioning Problem. In *Proceedings of International Symposium on Physical Design*, pages 192–198. ACM Press, 2000a.

[32] Ouyang, M., Toulouse, M., Thulasiraman, K., Glover, F., and Deogun, J.S. Multi-level Cooperative Search for the Circuit/Hypergraph Partitioning Problem. *IEEE Transactions on Computer-Aided Design*, 2000b. To appear.

[33] Rego, C. and Roucairol, C. A Parallel Tabu Search Algorithm Using Ejection Chains for the VRP. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory & Applications*, pages 253–295. Kluwer, Norwell, MA, 1996.

[34] Schulze, J. and Fahle, T. A parallel algorithm for the vehicle routing problem with time window constraints. *Annals of Operations Reseach*, 86:585–607, 1999.

[35] Taillard, É.D. Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[36] Taillard, É.D., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31:170–186, 1997.

[37] M. Toulouse. *Heuristiques parallèles de recherche sans contrôle global explicite*. PhD thesis, École Polythecnique, Université de Montréal, Montréal, Canada, 1996.

[38] Toulouse, M., Crainic, T.G., and Gendreau, M. Communication Issues in Designing Cooperative Multi Thread Parallel Searches. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory & Applications*, pages 501–522. Kluwer, Norwell, MA, 1996.

[39] Toulouse, M., Crainic, T.G., and Sansó, B. Systemic Behavior of Cooperative Search Algorithms. Publication CRT-97-55, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 1997.

[40] Toulouse, M., Crainic, T.G., and Sansó, B. An Experimental Study of Systemic Behavior of Cooperative Search Algorithms. In S. Voß, S. Martello, C. Roucairol, and Osman, I.H., editors, *Meta-Heuristics 98: Theory & Applications*, pages 373–392. Kluwer, Norwell, MA, 1999.

[41] Toulouse, M., Crainic, T.G., Sansó, B., and Thulasiraman, K. Self-Organization in Cooperative Search Algorithms. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2379–2385. Omnipress, 1998.

[42] Toulouse, M., Crainic, T.G., and Thulasiraman, K. Global Optimization Properties of Parallel Cooperative Search Algorithms: A Simulation Study. *Parallel Computing*, 26(1):91–112, 2000.

[43] Toulouse, M., Thulasiraman, K., and Glover, F. Multi-Level Cooperative Search. In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud, and D. Ruiz, editors, *5th International Euro-Par Parallel Processing Conference*, volume 1685 of *Lecture Notes in Computer Science*, pages 533–542. Springer-Verlag, Berlin, 1999.